Diseño e implementación del software del UPMSat-2 en el entorno de desarrollo TASTE

Jorge Garrido, Juan A. de la Puente, Juan Zamorano y Alejando Alonso
Grupo de Sistemas de Tiempo Real y Arquitectura de Servicios Telemáticos (STRAST)
Universidad Politécnica de Madrid (UPM)
str@dit.upm.es

***Abstract*— Como respuesta a la necesidad de modernizar y homogeneizar el proceso de diseño y desarrollo de software para el segmento de vuelo de sus misiones, la Agencia Espacial Europea puso en marcha en 2004 el proyecto ASSERT. El resultado de este proyecto fue una nueva metodología basada en el desarrollo basado en modelo. Posteriormente, la propia Agencia promovió un nuevo proyecto, TASTE, con el objetivo de desarrollar un entorno de desarrollo que permitiera la puesta en práctica de la metodología propuesta en ASSERT. En el presente artículo se describen las principales características de este entorno de desarrollo, así como la experiencia en su uso en el ámbito del proyecto UPMSat-2. (*Abstract*)**

Keywords—sistemas de tiempo real; desarrollo basado en modelos; TASTE; UPMSat-2;

I. INTRODUCCIÓN

El proyecto ASSERT (Automated proof-based System and Software Engineering for Real-Time systems), liderado por la Agencia Espacial Europea y formado por un consorcio de 28 socios del sector aeroespacial tuvo como principal objetivo el desarrollo de métodos más fiables y robustos para el desarrollo de sistemas de software embarcado. Esta metodología desarrollada se conoce como *ASSERT process*, siendo un conjunto de métodos y directrices con los que abordar el desarrollo, dirigido por modelos, de software embarcado que asegure la validez del mismo de acuerdo a sus requisitos.

Fruto del proyecto ASSERT surge el proyecto TASTE (The Assert Set of Tools for Engineering) [1,2], también liderado por la Agencia Espacial Europea, con el objetivo de generar un conjunto de herramientas de código abierto para el desarrollo de sistemas embarcados de tiempo real siguiendo la metodología propuesta en ASSERT. En el presente artículo se presenta la experiencia de uso de la herramienta en el diseño e implementación del software del satélite UPMSat-2.

El resto del artículo está estructurado de la siguiente manera: en la sección II se presenta brevemente el proyecto del satélite, mientras que en la sección III se presenta el mencionado conjunto de herramientas de TASTE. En la sección IV se presenta el proceso de diseño del software del satélite haciendo uso de las diferentes herramientas de TASTE. La sección V se centra en el proceso de implementación y compilación del código generado para su ejecución en la plataforma objetivo. Finalmente se presentan las conclusiones del uso de TASTE en el ámbito del proyecto del UPMSat-2.

II. UPMSAT-2

El UPMSat-2 es un proyecto de la Universidad Politécnica de Madrid que tiene el objetivo de desarrollar un micro-satélite experimental. Su función es la de servir como demostrador tecnológico de las capacidades técnicas de los diversos grupos involucrados en el proyecto. El Instituto Universitario de Microgravedad "Ignacio da Riva" lidera el proyecto. El Grupo de Sistemas de Tiempo Real y Arquitectura de Servicios Telemáticos (STRAST) es el encargado del diseño e implementación del software, tanto del segmento de tierra como del segmento de vuelo. El presente artículo describe el uso de TASTE en el desarrollo del software de vuelo o software embarcado.

A. Plataforma hardware

El software embarcado del satélite ejecutará sobre un único computador, conocido como On-Board Computer (OBC). El hardware del mismo está basado en la familia de computadores de 32 bits resistentes a la radiación LEON3 [3], que es una implementación de la arquitectura SPARC v8 [5]. El OBC incluye además memoria del tipo SRAM, y de tipo EEPROM para almacenamiento no volátil, así como diversos dispositivos periféricos para la interacción con sensores y actuadores.

La versión de ingeniería de la OBC se está desarrollando a partir de *IP cores* codificados en VHDL de procesadores LEON. Los *IP cores* del procesador LEON3 y de otros dispositivos se han sintetizado en una FPGA que constituye la base de la versión de ingeniería en la cual se están probando los diferentes elementos tanto software como hardware. A partir de esta versión de ingeniería se está desarrollando la versión de vuelo del OBC que contendrá chips de memoria y FPGA resistentes a la radiación.

B. *Arquitectura software*

El software de a bordo está compuesto por los componentes mostrados en la figura 1. Éstos son:

- Monitor de la plataforma: este subsistema se encarga de controlar el estado de los diferentes componentes del satélite mediante comprobaciones periódicas: medida de los voltajes y corriente de las baterías y paneles solares, la temperatura en diferentes puntos de la estructura y componentes del satélite.

- Telemetría y telecomando (TMTC): subsistema encargado de interactuar con el equipo hardware de telecomunicaciones para el envío de mensajes (telemetría) y recepción de órdenes (telecomando) de la estación de tierra.

- Sistema de determinación y control de actitud (ADCS): subsistema encargado de analizar la orientación actual del satélite respecto a la tierra y calcular la actuación para mantenerlo en los parámetros consignados. Este proceso se lleva a cabo mediante el procesado de las medidas del campo magnético de la Tierra en los tres ejes del satélite provistas por unos sensores específicos conocidos como magnetómetros (MM). Los correspondientes actuadores de este subsistema son los magnetopares (MT), electroimanes que generan un campo magnético que interactúa con el de la Tierra produciendo un par que gira al satélite sobre sus ejes.

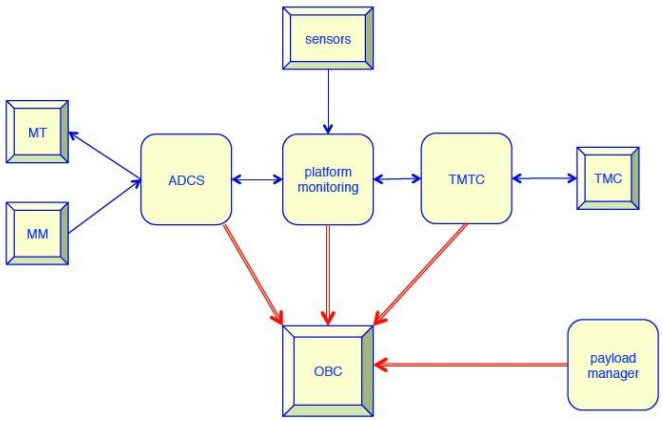

Figura 1 – Arquitectura software del satélite UPMSat-2.

- Controlador de experimentos: subsistema encargado de gestionar la puesta en marcha, ejecución, toma de datos y detención de los diversos experimentos a bordo del satélite, que constituyen su carga de pago o *payload*.

III. EL ENTORNO TASTE

TASTE es un conjunto de herramientas de código abierto para el desarrollo de sistemas empotrados y de tiempo real. Estos sistemas suelen tener requisitos de tiempo real crítico y, como parte de su validación, hay que realizar un análisis de planificabilidad para demostrar que tendrán un comportamiento temporal correcto. TASTE incluye un conjunto de restricciones que aseguran que los sistemas desarrollados son compatibles con el perfil de Ravenscar [9,10]. El perfil de Ravenscar es un conjunto de restricciones a la parte concurrente del lenguaje de programación Ada, a fin de que el código generado sea analizable temporalmente. Aunque se definió originalmente para el lenguaje Ada, su modelo computacional se puede extrapolar a otros paradigmas de programación concurrente como POSIX.

El conjunto de herramientas TASTE está enfocado al desarrollo basado en modelos [6], siguiendo la metodología desarrollada en el proyecto ASSERT. En ella se describen las siguientes fases: una primera fase de modelado del sistema, una posterior fase de transformación del sistema modelado a una arquitectura software de tiempo real, seguida de un análisis de factibilidad del sistema. Una vez superado este análisis, se pasa a la fase de generación de código, compilado y finalmente una fase de validación de la implementación del sistema. En las siguientes subsecciones se presentan en mayor detalle cada una de estas fases y las tecnologías implicadas en cada una de ellas.

A. *Fase de modelado*

La primera fase del desarrollo propuesta por TASTE es el modelado del sistema. El objetivo de esta fase es generar un modelo homogéneo para el sistema en su conjunto, obviando la naturaleza heterogénea del hardware y los artificios software necesarios para la implementación del mismo. El proceso de modelado por tanto debe basarse únicamente en una arquitectura software preliminar como la presentada en la figura 1, fruto del análisis conjunto del sistema objetivo por parte de los ingenieros del sistema y expertos en software.

Esta fase del proceso es posiblemente la más compleja. Por ello, se hace uso de diversos lenguajes como se presentará a continuación.

El nexo de unión entre los diferentes lenguajes usados en el desarrollo es otro lenguaje, AADL (Architecture Analysis and Design Language), un lenguaje de descripción de arquitecturas estandarizado por la Sociedad de Ingenieros de Automoción (SAE). AADL está definido por un núcleo del lenguaje que define una notación única tanto para características de hardware, software y comportamiento del sistema. AADL presenta una gran flexibilidad a la hora de modelar sistemas concretos, al poderse declarar características específicas de cada sistema usando propiedades (*properties* en el lenguaje).

Además, AADL se puede extender mediante dos métodos: mediante la extensión del conjunto de propiedades del sistema con propiedades definidas por el usuario y mediante la adición de anexos al lenguaje. Por el momento existen cuatro anexos, todos ellos relevantes en el ámbito de los sistemas que se pretende desarrollar con TASTE: *Behaviour annex*, que incluye soporte para máquinas de estados, el *Error-model annex*, que especifica diversos aspectos sobre el tratamiento y propagación de errores, el ARINC653 *annex*, que define patrones de modelado para sistemas de aviónica, y el *Data-Model annex* para el modelado de tipos de datos mediante AADL. Sin embargo, el conocimiento del lenguaje AADL no es necesario para el usuario de TASTE, ya que, aunque es posible hacerlo, no es necesario manipularlo directamente. Por contra, las diferentes herramientas de modelado transforman automáticamente a AADL las características del sistema especificadas por el usuario a medida que se editan los diferentes modelos o vistas.

Las vistas son, en el conjunto de herramientas de TASTE, cada una de las abstracciones para el modelado del sistema. Existen cuatro: *Data view*, *Interface view*, *Deployment view* y *Concurrency view*.

- *Data view*: en un primer paso se propone modelar los tipos de datos a usar en el sistema. Si bien sólo se exige estrictamente el modelado de aquellos datos que se vayan a usar posteriormente como parámetros en las diferentes interfaces, resulta altamente recomendable la especificación temprana de todos aquellos tipos a usar relacionados con la lógica del sistema. Esta especificación es abstracta y no impide que cada módulo de software use los tipos propios del lenguaje en el que finalmente se implemente.

  El lenguaje seleccionado en TASTE para llevar a cabo este proceso de modelado fue ASN.1 [7]. ASN.1 es un lenguaje estandarizado por ISO y la ITU-T que permite el modelado de tipos y estructuras de datos, tanto desde el punto de vista semántico como de codificación. Las especificaciones generadas mediante ASN.1 son independientes de los lenguajes de programación, de las plataformas hardware e incluso de la codificación de las estructuras de datos definidas. ASN.1 se encarga por defecto de generar una codificación óptima a los tipos especificados. Sin embargo, también es posible, mediante la extensión ACN, permitir a los diseñadores definir el formato de codificación a nivel de bit. Esta característica resulta de alto interés en este tipo de sistemas, en especial para el manejo de las interfaces con sensores y actuadores.

- *Interface view*: en un segundo paso se propone el modelado de las interfaces software mediante la vista de interfaz. Haciendo uso de una herramienta gráfica, se definen las diferentes funciones del sistema. Estas funciones pueden ser agrupadas en contenedores, que si bien no representan ninguna información efectiva, hacen más sencilla e intuitiva la edición del modelo. Las interacciones entre funciones se modelan mediante interfaces provistas y requeridas. Estas últimas se caracterizan por un conjunto de propiedades no funcionales (entre ellas los requisitos temporales de la tarea). Este conjunto de propiedades no funcionales depende del tipo de tarea que representen. Así, por ejemplo, en el caso de tareas cíclicas se ha de definir su periodo, mientras que el caso de tareas esporádicas se debe dar un valor de tiempo mínimo entre llegadas.

  Del mismo modo que en el caso de la edición del modelo de datos, el modelo se convierte a AADL siendo accesible para uso en el resto de herramientas del entorno.

- *Deployment view:* Una vez definidas las diferentes funciones de software y las dependencias entre ellas se debe proceder a describir la arquitectura hardware del sistema. Mediante una herramienta gráfica similar a la utilizada en el paso anterior se despliegan diferentes elementos sobre el modelo. Los dos elementos básicos de este modelo son los *Processor Boards* y los *buses* que interconectan a los primeros entre sí. Cada *processor board* tiene al menos un procesador (caracterizado por su arquitectura) y un módulo de memoria principal. Adicionalmente, un *processor board* puede tener más de un módulo de procesador o memoria, además de varios módulos de *Drivers* mediante los cuales se conecta a los diferentes *buses*. Dentro de cada procesador se define al menos una partición, caracterizada por el sistema operativo o *kernel* que se ejecutará sobre ella. Finalmente el usuario asigna las funciones software declaradas en la *Interface view* a la partición en que se desplegarán.

- *Concurrency view:* analizando automáticamente los modelos generados tanto en el *Interface view* como el *Deployment view*, TASTE genera una nueva especificación AADL que recoge las características de concurrencia y rendimiento del sistema, pudiendo analizar la planificabilidad del mismo mediante la herramienta integrada Cheddar[1].

B. *Generación de código*

Una vez modelados los diferentes módulos o funciones software del sistema en la *Interface view*, TASTE permite generar automáticamente tanto los esqueletos de código de cada función como los *scripts* necesarios para su posterior compilación. De igual modo, genera las definiciones de los tipos declaradas en el *Data view* para cada uno de los diferentes

[1] http://beru.univ-brest.fr/~singhoff/cheddar/

lenguajes soportados por TASTE. Estos lenguajes son los siguientes:

- SDL (Specification Description Language)[8], lenguaje de especificación de sistemas complejos mediante la comunicación entre módulos independientes por medio de señales. SDL está estandarizado por la ITU-T en el estándar Z.100.
- Simulink[2], un entorno de modelado, simulación y análisis de sistemas dinámicos. TASTE permite la implementación de funciones mediante modelos generados con esta herramienta.
- Ada [11], lenguaje de programación orientado a objetos y fuertemente tipado, usado principalmente en el entorno de los sistemas de tiempo real y sistemas embebidos.
- C, lenguaje de programación de ámbito general.
- SCADE[3], herramienta de diseño y modelado de aplicaciones de alta criticidad para sistemas embebidos.

Además de estos lenguajes para la implementación de software, las funciones de la *Interface view* pueden ser declaradas con otros lenguajes. Uno de ellos es VHDL, permitiendo dentro del propio entorno definir elementos hardware en este lenguaje, integrados con el resto del sistema.

C. Compilación

Mediante los *scripts* generados automáticamente por TASTE se puede compilar el código de las diferentes funciones con los compiladores incluidos en la herramienta y enlazarlos para producir los ejecutables, uno por cada partición declarada en el *Deployment view*.

En esta fase, se hace uso de un software de intermediación (middleware) específico, conocido como PolyORB-HI encargado de relacionar las diferentes primitivas del código generado a aquellas ofrecidas por el sistema operativo seleccionado para cada partición. Este software de intermediación se ofrece en dos variantes: una en Ada, que se implementa mediante un *run-time system* que da soporte al perfil de Ravenscar [4] y por tanto el compilador y el *run-time* realizan comprobaciones del código generado para asegurar que se cumplen las restricciones, y otra variante en C/POSIX, en el cual las restricciones las comprueba el propio PolyORB-HI. El uso de una u otra variante depende de los requisitos específicos de cada proyecto en términos de uso de memoria, rendimiento, disponibilidad de drivers, etc.

[2] http://www.mathworks.es/products/simulink/
[3] http://www.esterel-technologies.com/products/

D. Validación

TASTE ofrece diferentes herramientas para la validación del sistema generado. En primer lugar, TASTE puede generar automáticamente pruebas de codificación y decodificación de los diferentes tipos definidos. Dado que a lo largo del sistema un mismo valor puede pasar por elementos software desarrollados en diferentes lenguajes, y por elementos hardware con diferentes *endianness*, esta característica resulta de gran utilidad.

Por su parte, en el ámbito de la *Interface view* se pueden declarar funciones para definir un interfaz gráfico. Estas funciones generan un ejecutable adicional, consistente en una interfaz gráfica, en la cuál se pueden tanto mostrar los valores recibidos por las interfaces ofrecidas, como enviar diferentes valores por las interfaces requeridas. En esta interfaz gráfica se puede, además almacenar las interacciones con el sistema, generando un conjunto de pruebas automático basado en las pruebas realizadas previamente de forma manual, característica de utilidad para la realización de pruebas de regresión. El uso habitual que se hace de esta funcionalidad es la de simular la estación de tierra, declarando una interfaz ofrecida para recibir la telemetría del satélite, y una interfaz requerida para enviar telecomandos al mismo. De esta manera se puede interactuar manualmente con el sistema en desarrollo y analizar su comportamiento. Un ejemplo de esta funcionalidad se presenta en la sección V.

Otra interesante característica para la validación del sistema es la disponibilidad de la herramienta *PeekPoke*. Mediante esta herramienta se puede monitorizar en tiempo de ejecución las diferentes variables del sistema. Esta herramienta, en conjunción con la presentada anteriormente forman una potente combinación para el análisis del comportamiento del sistema.

IV. Diseño del UPMSat-2 en TASTE

Para el diseño del satélite UPMSat-2 se está haciendo uso de las herramientas proporcionadas por el entorno TASTE. De este modo, el grupo cumple uno de los objetivos principales del proyecto: hacer uso de las metodologías y herramientas de referencia en el entorno aeroespacial. En esta sección se presentan diferentes secciones del diseño en cada una de las vistas propuestas por TASTE.

A. Modelado de datos

Como se describió anteriormente el primer paso en el diseño es el modelado de los diferentes tipos de datos. Para ilustrar este proceso, se presenta el modelado de la estructura de telecomandos estandarizada por la ECSS [12], descrita en las figuras 2 y 3.

Packet Header (48 Bits)							Packet Data Field (Variable)			
Packet ID				Packet Sequence Control		Packet Length	Data Field Header (Optional) (see Note 1)	Application Data	Spare	Packet Error Control (see Note 2)
Version Number (=0)	Type (=1)	Data Field Header Flag	Applica-tion Process ID	Sequence Flags	Sequence Count					
3	1	1	11	2	14					
16				16		16	Variable	Variable	Variable	16

Figura 2 – Estructura de paquetes de telecomando estandarizado por ECSS. Extraído de [12].

CCSDS Secondary Header Flag	TC Packet PUS Version Number	Ack	Service Type	Service Subtype	Source ID	Spare
Boolean (1 bit)	Enumerated (3 bits)	Enumerated(4 bits)	Enumerated (8 bits)	Enumerated (8 bits)	Enumerated (n bits)	Fixed BitString (n bits)
					← Optional →	← Optional →

Figura 3 – Detalle del *Packet Data Field* del paquetes de telecomando. Extraído de [12].

En primer lugar se debe especificar el modelo ASN.1. En el haremos uso tanto de tipos de datos básicos (booleanos y enteros), como estructuras de datos complejas. En este ejemplo requeriremos del tipo SEQUENCE de ASN.1, una colección ordenada de variables de diferente tipo, algo equivalente a lo que en C sería el tipo *struct*. También se hace uso del tipo CHOICE, una colección de tipos posibles para una variable, de los cuales se selecciona uno en tiempo de ejecución, generalmente en función del valor de otras variables. Como podemos ver en el código correspondiente a la estructura de los paquetes de telecomando, este tipo es de gran utilidad para modelar campos de tamaño variable, algo bastante corriente en la especificación de protocolos de comunicación. Así, en el ejemplo se ilustran dos posibles contenidos para el *Application Data* en función del subsistema del satélite al que está dirigido el telecomando.

```
-- Telecommand structure
TelecommandPacketType   ::= SEQUENCE {
                tcPacketHeader TcPacketHeaderType,
                tcPacketDataField TcPacketDataFieldType
}

-- Packet Header
TcPacketHeaderType  ::= SEQUENCE {
                tcPHHeaderID TcPHHeaderIDType,
                tcPHSecuenceHeader TcPHSecuenceHeaderType,
                tcPHPacketLength TcPHPacketLengthType
}

-- Packet ID
TcPHHeaderIDType    ::= SEQUENCE {
                tcPHVersionNumber TcPHVersionNumberType,
                tcPHType TcPHTypeType,
                tcPHDataFielHeaderFlag TcPHDataFielHeaderFlagType,
                tcPHApplicationProcessID TcPHApplicationProcessIDType
}

TcPHVersionNumberType ::= INTEGER (0..7)

TcPHTypeType ::= INTEGER (0..1)

TcPHDataFielHeaderFlagType ::= INTEGER (0..1)

TcPHApplicationProcessIDType ::= INTEGER (0..2047)

...

-- Packet Data Field
TcPacketDataFieldType ::= SEQUENCE {
                tcDFH TcDFHType,
                tcApplicationData TcApplicationDataType,
                tcSpare TcSpareType,
                tcPacketErrorControl TcPacketErrorControlType
}

-- Data Field Header
TcDFHType ::= SEQUENCE {
                ...
                Ack TcDFHTAckType
}
...
TcDFHTAckType ::= SEQUENCE {
                        acceptanceAck BOOLEAN,
                        startAck BOOLEAN,
                        progressAck BOOLEAN,
                        completionAck BOOLEAN
}
-- Application Data
TcApplicationDataType ::= CHOICE {
                        TMRequest TMRequestType,
                        ExperimentCommand ExperimentCommandType,
...
}
...
```

Listado 1 – Definición de la estructura de un paquete de telecomando mediante ASN.1.

Una vez definida la estructura del paquete, es conveniente en este caso completarla con directrices sobre su codificación. Dado que el protocolo define claramente el tamaño de cada campo, así como el significado de cada bit según su posición en el paquete, debemos asegurarnos que estos se codifican adecuadamente. En otros casos podemos simplemente confiar en la codificación automática del compilador de ASN.1, el cuál asegura que esta codificación hace uso del tamaño mínimo posible para cada tipo de dato declarado. Además, el conjunto de herramientas TASTE posee mecanismos para asegurar la consistencia de los valores que tomen cada tipo de datos, independientemente del lenguaje de programación y plataforma en los que se interpreten. En los casos en los que sea necesario, se pueden dar directrices de codificación para cada tipo de datos definido en el modelo ASN.1 mediante ACN. Estas directrices indican los valores deseados para el tamaño de los datos, el tipo de representación, y en el caso de tipos de tamaño múltiplo de byte si se desea representación *little endian* o *big endian*, entre otros. A continuación podemos observar un extracto del código ACN generado para el paquete de telecomandos.

```
...
  TcPHVersionNumberType[size 3, encoding pos-int]
  TcPHTypeType[size 1, encoding pos-int]
  TcPHDataFielHeaderFlagType[size 1, encoding pos-int]
  TcPHApplicationProcessIDType[size 11, encoding pos-int]
...
```

Listado 2 – Directivas de codificación en ACN.

B. Modelado de interfaces

En lo que al modelado de interfaces se refiere, en la figura 4 se presenta la vista de interfaces de TASTE. Se puede observar que las funciones se han agrupado en contenedores equivalentes a los subsistemas presentados en la figura 1. Dentro de cada contenedor se han declarado las diferentes funciones requeridas por cada uno de los subsistemas. También se incluyen las interfaces provistas (flechas que apuntan a la función) y requeridas (flechas salientes de la función) así como las relaciones entre ellas.

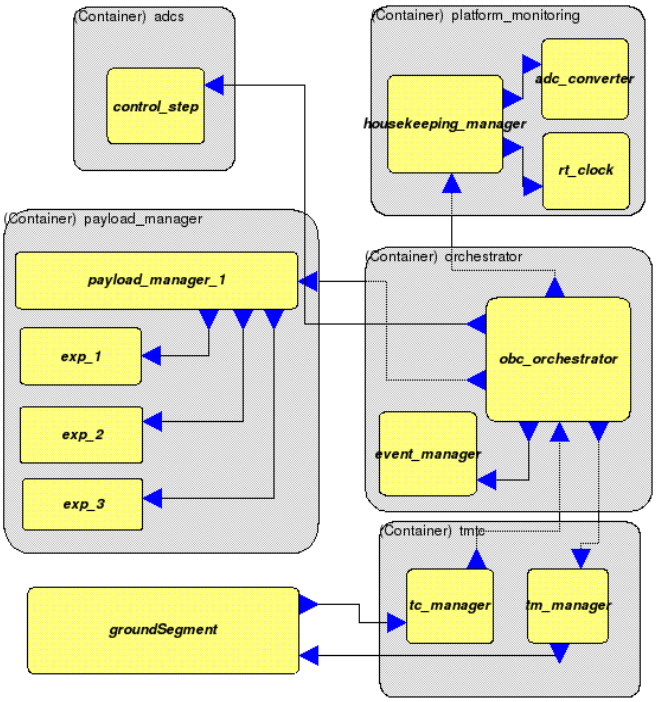

Figura 4 – Modelo de interfaces del UPMSat-2 en TASTE.

Para la validación se ha incluido la función *groundSegment*. Esta función reemplaza en el modelo de pruebas el sistema de enlace de comunicaciones entre el satélite y el segmento de tierra, de manera que en ambos segmentos se trabaja a nivel de aplicación. En posteriores refinamientos del modelo se deben incluir pruebas que cubran estos niveles.

C. Despliegue de interfaces

Finalmente se modela la vista de despliegue. En este caso es simple porque el satélite tiene un único OBC y por tanto todas las funciones declaradas en el *Interface view* se asignan a una partición sobre el único procesador disponible, como se puede ver en la figura 5.

V. IMPLEMENTACIÓN Y VALIDACION DEL UPMSAT-2 EN TASTE

Una vez generadas las interfaces, el entorno TASTE permite generar los esqueletos de las diferentes funciones, agilizando el desarrollo de la implementación. Las funciones se agrupan en particiones según se haya especificado en la *Deployment view*. Por último, el entorno genera los *scripts* necesarios para la compilación cruzada de todas las particiones.

Como se avanzó anteriormente, en esta fase se pueden desarrollar cada una de las funciones, no sólo en lenguajes de programación convencionales, sino también en otros lenguajes orientados a modelos (como Simulink) y máquinas de estados en SDL. Un ejemplo parcial del controlador de experimentos se presenta en la figura 6.

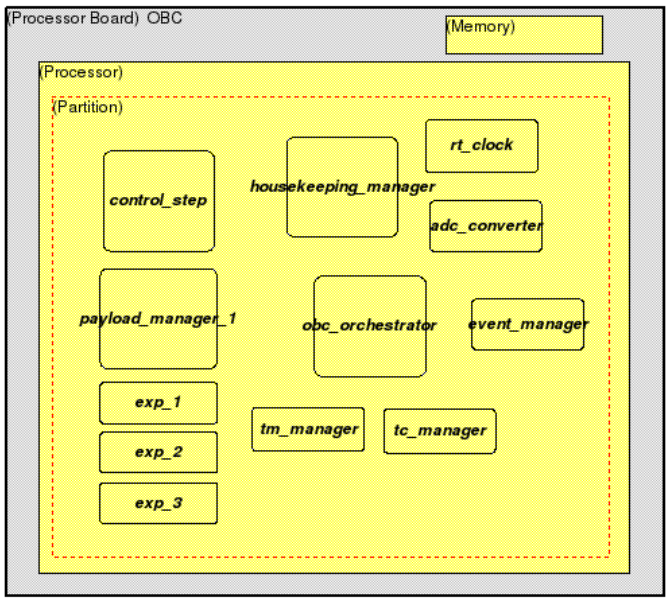

Figura 5 – Modelo de despliegue del UPMSat-2 en TASTE.

Para una primera validación de parte del software de alto nivel, se ha hecho uso de la función GUI declarada previamente en la *Interface view*. Mediante esta herramienta se envían diferentes telecomandos y se comprueban los valores recibidos en la telemetría. En el ejemplo mostrado en la figura 7 se muestra la petición del valor del reloj de misión por parte de la estación de tierra y la respuesta del satélite.

VI. CONCLUSIONES

El entorno de desarrollo TASTE ofrece notables facilidades para el desarrollo de sistemas de tiempo real empotrados. Desde la fase de diseño a la validación del sistema, el entorno integra un conjunto de herramientas de código abierto específicas para el diseño de este tipo de sistemas. De igual modo, los lenguajes empleados en cada una de las fases están estandarizados por diversos organismos, permitiendo al usuario

final hacer uso de herramientas externas para el desarrollo de cualquiera de las fases.

En el ámbito del satélite UPMSat-2 no sólo supone una interesante opción a la hora de implementar el sistema, sino también para iterativamente diseñar el mismo mediante las herramientas ofrecidas. Dada la metodología propuesta por TASTE, resulta sencillo ampliar o modificar cualquiera de las partes del diseño o implementación, adaptándose a los requisitos del proyecto.

El conjunto de herramientas responde a una metodología especialmente desarrollada para el desarrollo de sistemas de tiempo real empotrados basada en modelos. Esta metodología, si bien quedó definida en el desarrollo del proyecto ASSERT, se encuentra en proceso de revisión y ampliación, con el objetivo de ofrecer soporte al análisis de requisitos y mejorar la trazabilidad y verificación de los mismos, así como el mantenimiento del sistema, completando así el ciclo de vida del mismo.

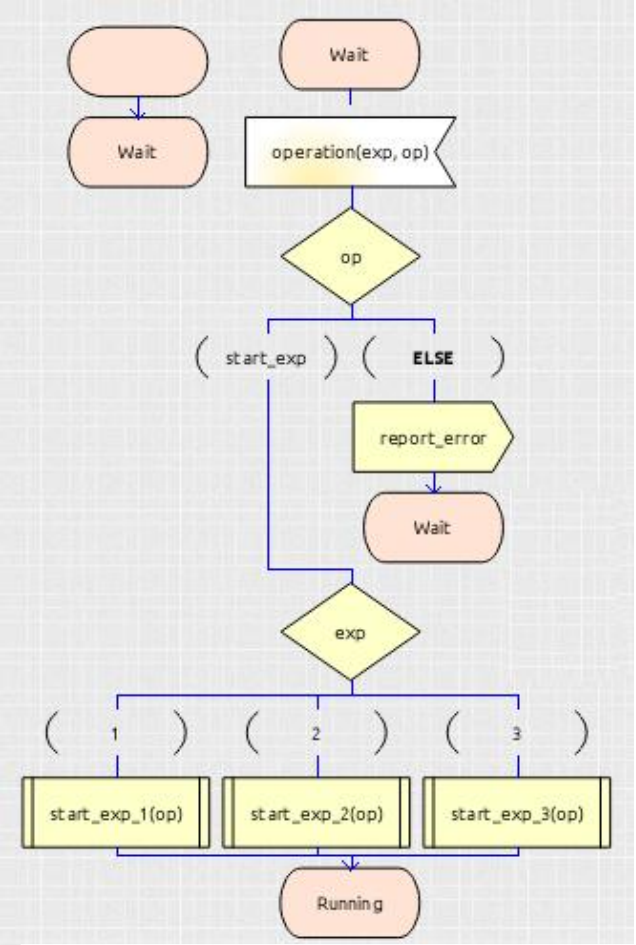

Figura 6 – Implementación parcial del controlador de experimentos como una máquina de estados.

Referencias

[1] de la Puente, Juan Antonio, et al. "The ASSERT Virtual Machine: A predictable platform for real-time systems." World Congress. Vol. 17. No. 1. 2008.

[2] Perrotin, Maxime, et al. "TASTE: a real-time software engineering tool-chain overview, status, and future." SDL 2011: Integrating System and Software Modeling. Springer Berlin Heidelberg, 2012. 26-37.

[3] Gaisler Research. "LEON3 - High-performance SPARC V8 32-bit Processor." GRLIB IP Core User's Manual, 2012.

[4] Juan, A., José F. Ruiz, and Juan Zamorano. "An open Ravenscar real-time kernel for GNAT." Reliable Software Technologies Ada-Europe 2000. Springer Berlin Heidelberg, 2000. 5-15.

[5] SPARC International. "The SPARC architecture manual." SPARC International, Upper Saddle River, NJ, USA.: Version 8, 1992.

[6] D. C. Schmidt. "Model-driven engineering." IEEE Computer, 39(2), 2006.

[7] Dubuisson, Olivier. "ASN. 1: communication between heterogeneous systems." Morgan Kaufmann Pub, 2001.

[8] Ellsberger, Jan, Dieter Hogrefe, and Amardeo Sarma. "SDL: formal object-oriented language for communicating systems." Prentice Hall, 1997.

[9] Burns, Alan, Brian Dobbing, and Tullio Vardanega. "Guide for the use of the Ada Ravenscar Profile in high integrity systems." ACM SIGAda Ada Letters 24.2 (2004): 1-74.

[10] Mezzetti, Enrico, Marco Panunzio, and Tullio Vardanega. "Preservation of timing properties with the ada ravenscar profile." Reliable Software Technology, Ada-Europe 2010. Springer Berlin Heidelberg, 2010. 153-166.

[11] ISO. "Ada Reference Manual ISO/IEC 8652:1995(E)/TC1(2000)/AMD1(2007)," 2007. Available at http://www.adaic.com/standards/ada05.html.

[12] European Cooperation for Space Standardization. "ECSS-E-70-41A Space engineering — Ground systems and operations – Telemetry and telecommand packet utilization", January 2013. Available from ESA.

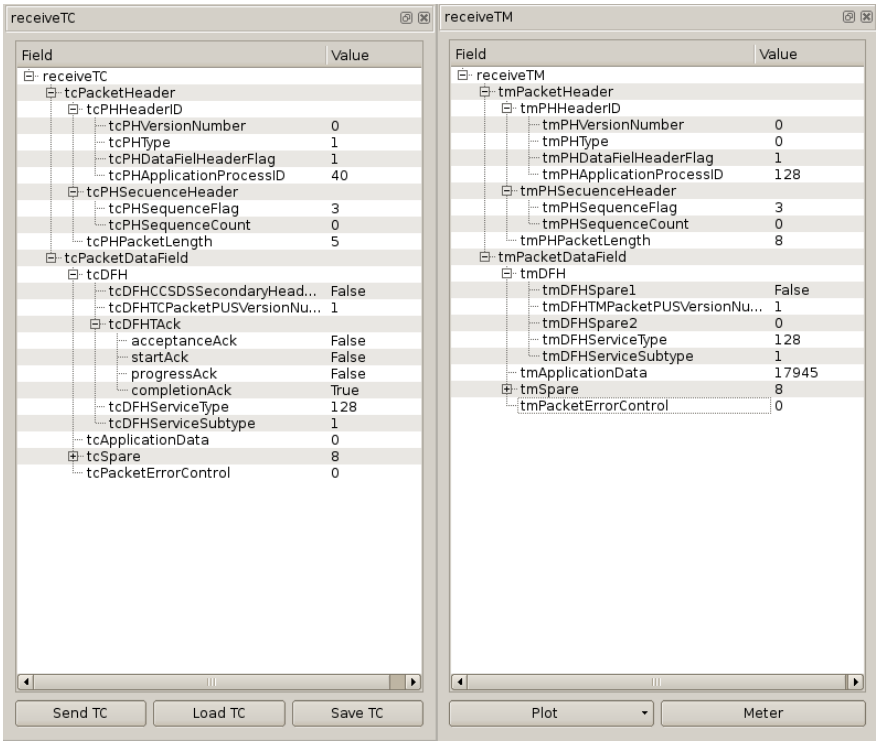

Figura 7 – La herramienta gráfica permite de forma sencilla analizar la respuesta del satélite a las diferentes órdenes enviadas desde el segmento de tierra.